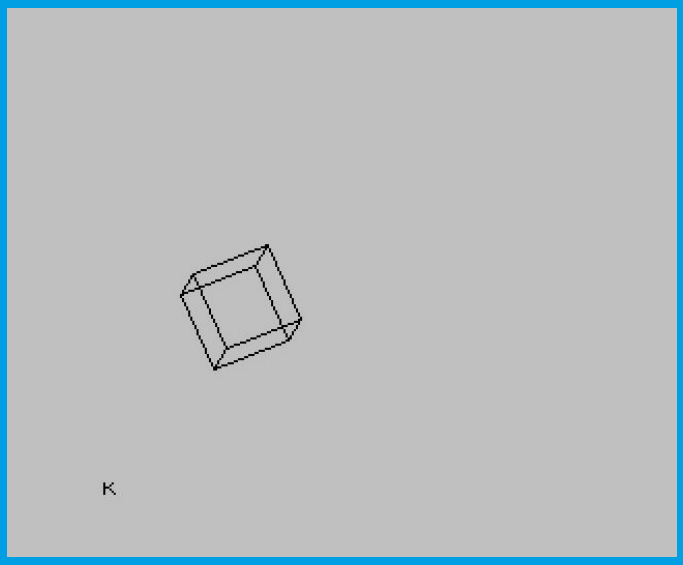
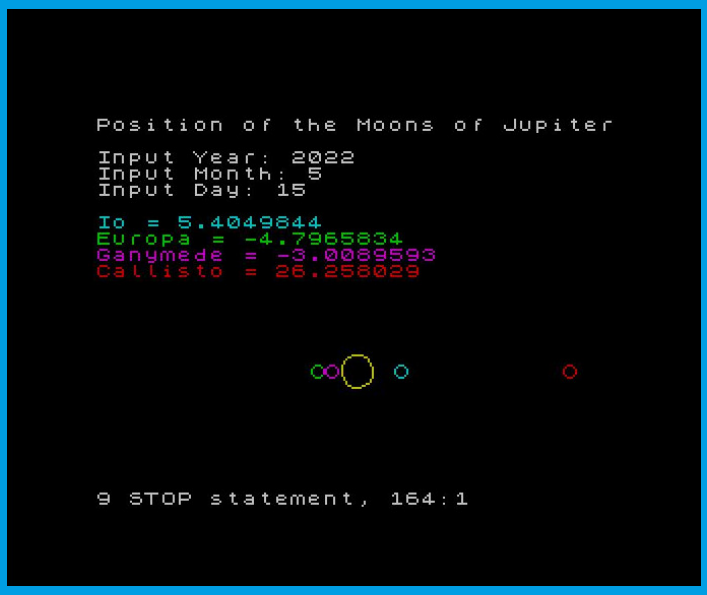
2x2ine

Issue #9 **June 2022**



Wireframe Graphics



Locations of the Moons of Jupiter

New Fonts for ZX81

ZXzine

Table of Contents

Published by:
Timothy Swenson
swenson_t@sbcglobal.net
swensont@lanset.com

ZXzine is published as a service to the Sinclair ZX81 community. Writers are invited to submit articles for publication. Readers are invited to submit article ideas.

Created using Open Source Tools:

- OpenOffice
- Scribus
- Gimp
- SZ81
- Zesarux

Editorial

I've been having Zoom meeting with other T/S 2068 enthusiasts. David Anderson, who hosts www.timexsinclair.org, started the meetings. He's been beating the drum of preservation to the team. He's looking for Timex/Sinclair books, programs, newsletters, etc. to preserve. Some of the meeting attendees have come up with some good documents.

Since I was a member of a number of T/S user groups, I have a couple of binders full of newsletters. A lot from the Capital Area Timex/ Sinclair (CATS) user Group and TimeLinez, which was the newsletter for the three SF Bay Area user groups, East Bay, Peninsula and Silicon Valley.

I have spent a number of hours scanning in these newsletters. I've also scanned a few other sorted newsletters that I have. I had previously scanned some International QL Report (IQLR) newsletters.

All of this David has converted to PDF files and posted to the Internet Archive. I always knew that I was keeping these newsletters to eventually preserve them. It is good to get it done and now others can read these newsletters.

I never really had much T/S 2068 software, so I can't help on the preservation of software.

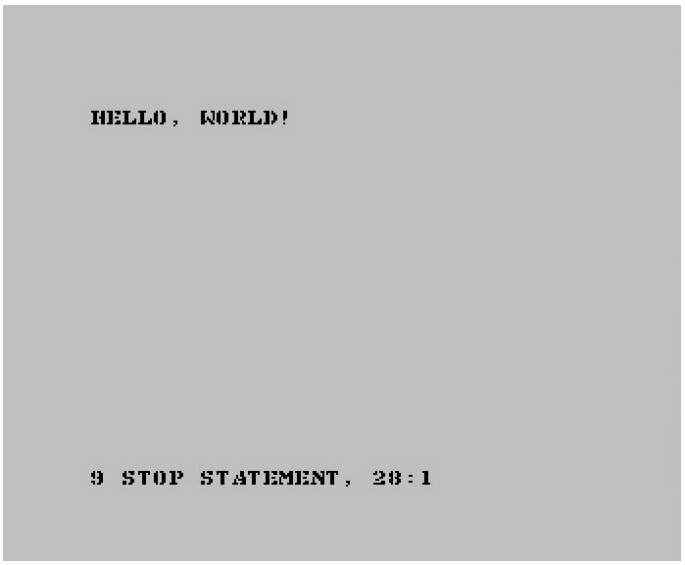
This issue has a few articles based on the material I found while scanning the newsletters. I hope it is interesting to some.

T/S 2068 Fonts

One feature of the T/S 2068 that I had never tinkered with is using a different font. I knew a number of Spectrum games that used a different font, but I had not thought about it on the T/S 2068. The general process is the same on the T/S 2068 as it is on the Spectrum.

In most cases, the font is loaded from tape and then used until the computer is powered off. I did not want to do it that way. I wanted to have the font embedded into the program, be it a BASIC program or an assembly language program. To do this I needed a font to be in numbers for a DATA statement and not a binary file.

I did some looking online and found the website of Jim Blimey. He found some fonts online and had converted them for use on the Spectrum. They came in a number of formats, one being a text file with the fonts in defb statements for use with assembly programs. There was also an image (png) file of each font so it was easy to see what they looked like.



Jim has two font zip files; 24-zxspectrum-fonts.zip from a user called EPTO on Github, and charbank.zip from the "Load'n Run" magazine from Italy.

To test with, I picked one of the fonts from the first zip file. I used an editor to change the "defb" text to "DATA". I then wrote a short program that loaded the font from the DATA statements into memory. Next was the POKE to the CHARS system variable to point the system to use the new font. The last bit of the code was to print "Hello, World" in the new font. I processed the program through zmakebas and loaded the .tap file into Zesarux. The program ran as expected and showed a new font on the screen.

The next part of the project was to do the same using assembly. As I was writing a loop in assembly that would copy the font from the defb statements to memory, I was stumbling on what registers to use. I needed a pair for keeping track of

the location of the font and another par to keep track of the destination of the font, and another register to keep track of how many bytes is being moved.

As I was breaking out the Z80 book by Lance Leventhal, I ran across the LDIR command. Instead of writing a loop, the LDIR command does all of the work. I just needed to load into HL where the font was coming from, into DE where the font was going to and into BC the number of bytes to move. LDIR does the copy, increments both DE and HL, and decrements BC. The whole process exits when BC = 0.

Initially my program was not working. It seems to work, but when I did the POKE to set CHARS, the system crashed. I was thinking there was something wrong with the POKE, but it turns out I had swapped the purpose of DE and HL. Once I had that straightened out, the program worked.

The one downside of my approach is that the font will take up double the normal space for the font. There will be the font in the source code and the font in memory. Each font is 768 bytes, so with a 48K system, I'm fine with the extra cost in memory.

The values used in fonts is basically the same as used in User Designed Graphics (UDG). Users can create up to 21 UDG's, but if you need to create more, it is possible to create a new font with a number of the characters to be functionally UDG's.

I was thinking that I might do something like that, but I did not want to use a new font and add what I needed, but I wanted to use the default font. I wrote a program to go through the 96 characters in the default font (by reading the ROM) and show the decimal numbers for each 8 bytes of the font. I then put that into a text file with DATA statements. I used that font in my program to load a font just to test that I had pulled the default font from ROM correctly, that I had put it into a font file correctly and that the default font could be loaded correctly. Now I could edit the default font and replace any characters with ones of my own. That will be a project for another time.

Wireframe for T/S 2068

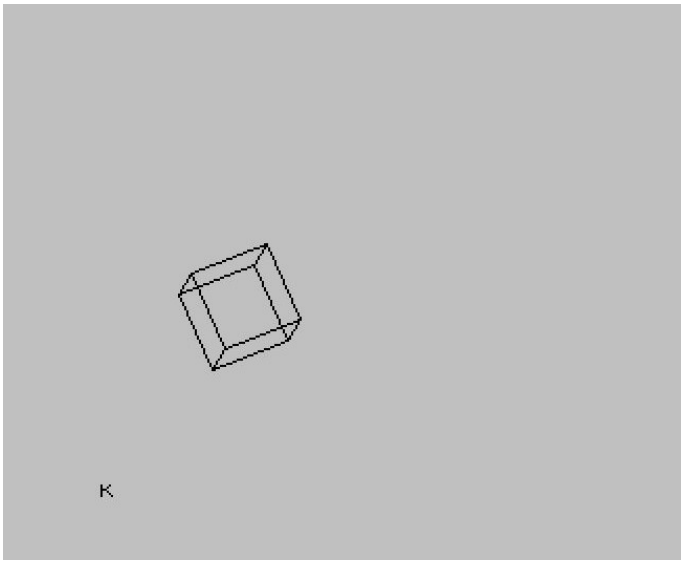
Back in college I took a Computer Graphics class. One of the items covered was wireframe 3D graphics (well, 3D showing in 2D). I wrote a program for the QL using the technique used in the book from that class. I decided to port the program over to the T/S 2068.

The port was fairly straight forward. There was nothing in the code that the T/S 2068 could not handle. When I ran the program, it failed. I then realized that the QL and T/S 2068 DRAW commands are far different. On the QL, DRAW is absolute, meaning that the start point and end point are the coordinates on the screen. With the T/S 2068, the points are relative. You have to first PLOT a point to set the start point, then the point used in the DRAW command is relative to the starting point. So, I just had to some a little conversion and the program worked.

The program is mostly composed of subroutines that are called to manipulate the 3D object. The user then writes their own code to call these routines. The wireframe.bas program has the subroutines and an example of "user code". You can change this "user code" section and do something different.

The subroutines form a wireframe library. The routines in the library are:

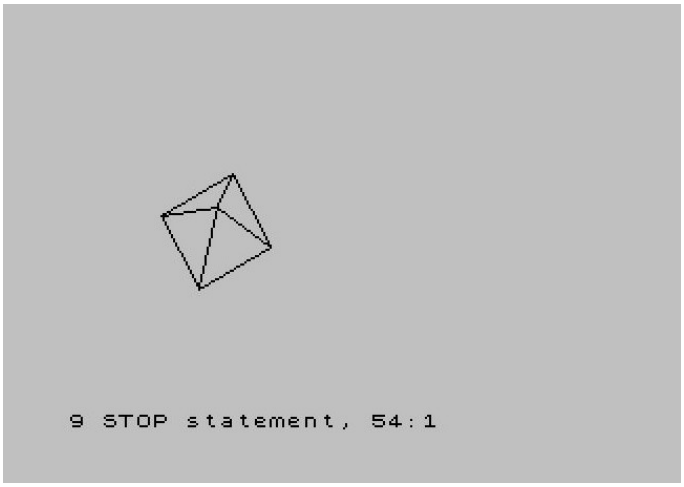
square - defines the data set for a cube.



pyramid - defines the data set for a pyramid.

tetra - defines the data set for a tetrahedron.

loaddata - once "square", "pyramid" or "tetra" are



called, then this subroutine is called to load the data into the arrays.

rotz - rotate the object in the Z axis.
roty - rotate the object in the Y axis.
rotx - rotate the object in the X axis.

enlarge - makes the object bigger in screen.
shrink - makes the object smaller on screen.
display - draws the object on screen.

On the QL these were procedures that can be called by name. On the T/S 2068 they are subroutines that can be called by name using labels with zmakebas. To use rotz, do this:

```
gosub @rotz
```

There are two additional subroutines included in the program:

initspin - does a rotation in all three axis to show off the object better on the screen.

spin - does a spin of 10 rotations in the Z direction.

Run the program to get a good idea of how things work and how quick the program is in drawing the object.

It is possible to define new objects. The two main data statements define the vertexes and the edges.

The vertexes are the 3D points that define the shape of the object. The edges define the lines between the vertexes. It is the lines that are drawn.

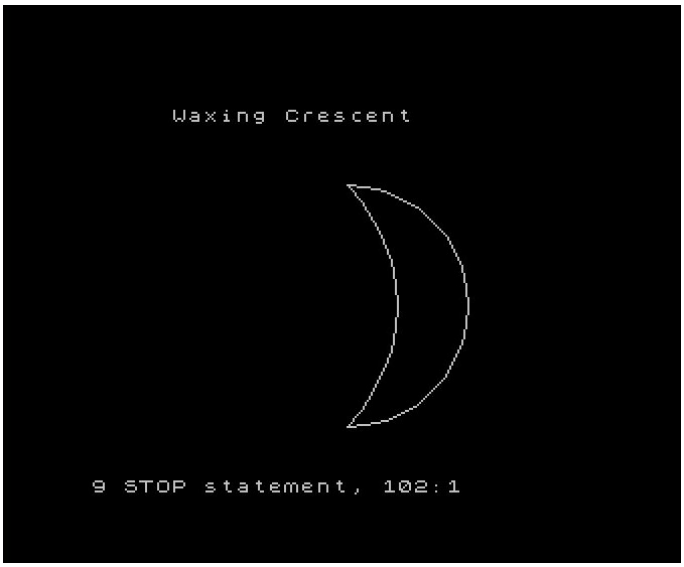
The first data in each line is the number of vertexes or edges. The rest of the data statement are those vertexes or edges. The first vertex is #1, the second #2, etc. Edges are defined as two vertexes and the vertexes are referenced by number.

The middle of each object is 0,0,0. With the cube (not sure why I called it a square in the code), it is 10 units on each side. That is why it is comprised of vertexes with 10 or -10.

Moonphase and Jupiter's Moons

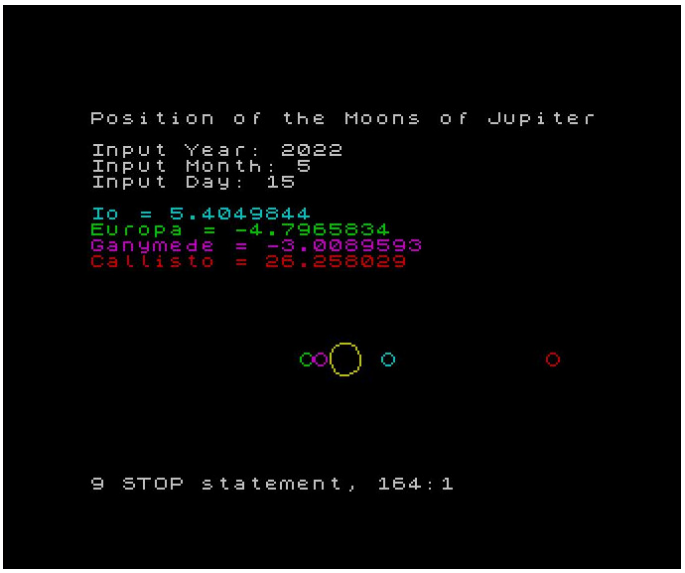
I've done a number of astronomy programs on the ZX81, but really have not ported any to the T/S 2068. I found two programs that would do well with a port and they would benefit from the better graphics on the T/S 2068.

Moonphase is a simple program that when given a date, will determine the phase of the moon for that date, from a range of 8 phases. Most of the moonphase programs generate the same 8 phases. The base program just prints out the phase based on



the date. With the T/S 2068 version, the phase of the moon is drawn on the screen.

Given how long it has been since I used the T/S 2068 arc drawing graphics, it took me a while to get the hang of it. I had to re-learn how to get the arc to



bend one way or the other. How to get the distance of the arc right. I even had to make sure which way was waxing and which way was waning.

Jupiter's Moons is a program that will determine the locations of the four main moons of Jupiter, based on the date. The result is the distance in Jupiter radians of the moon from Jupiter. The four major moons are Io, Europa, Ganymede and Callisto. The program takes some time to calculate the positions. Once done, it will print out the distance of each moon and then it will graph it in relation to Jupiter. A circle is drawn for Jupiter and the moons, with each being in a different color. If the moons are too close to each other or Jupiter, there can be some color clash with the most recent circle drawn over writing the earlier color.

UDG Character Generator

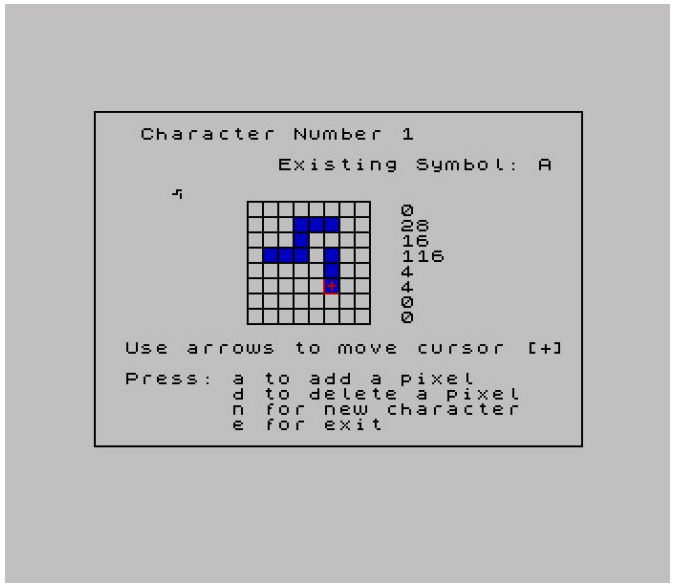
While scanning some TimeLinez newsletters, I came across an book review by Walt Gaby of the book "Timex Sinclair Color Graphics" by Nick Hampshire. In his review, Walt included a program that helps define the codes needed with User Defined Graphics (UDGs). Having touched on T/S 2068 fonts (which are similar to UDGs), I thought I would type in the program and see how well it functioned.

If I need one or two UDGs I found it simple to just get some graph paper, draw the UDG and then use a computer calculator that can convert binary to decimal. Of course, if I need to make changes during the design process I have to make sure I use a pencil so I can make changes. With this Character Editor, changes can easily be made.

The program draws a grid for the 8x8 pixels. The arrow keys are used to move around the grid. The "a" and "d" keys are used for setting a pixel or unsetting a pixel. The original program used "A" and "D", but I change it to use lower case so that the shift key would not need to be used.

Once pixels are set, the value of each row is shown and the UDG up to that point is also shown.

There is no provision for saving a UDG. It is assumed that you will just write down the 8 decimals numbers that define that UDG and then use them in your programs.



Lunar Eclipses

While scanning some Capital Area Timex Sinclair (CATS) newsletters I ran across this program that will predict lunar eclipses for a given year. All though I found it in the CATS newsletter, the program was originally published in Sinc-Link newsletter by Mel Richardson. The original code was written by Herbert Raab of Austria, using different code bits published in "Sky & Telescope" magazine. Mel converted the program to run on Sinclair BASIC.

Liking astronomy programs, I had to type the program in and see it working. Once typed and all of my typo's corrected, I ran the program for the

example year, 1989. The results that I got matched what was published. That was proof that it was working as advertised.

Some astronomy programs are only good for a few given years. The further way from those years the less accurate the program is. To see if this program was like that, I ran it for 2022. We just had a lunar eclipse in the middle of May, so I'd see it it would catch it. I started the program up, entered 2022, waited a few minutes and there is the May eclipse. The date format is the European format of day, month, year. The times are in Universal Time, so one needs to convert to local time. The program gives the time of the peak of the eclipse. The information I had had only the start and stop times of the eclipse, but the given max time of the eclipse looked right.

```
ENTER YEAR:

ECLIPSE DATE:    16/5/2022
MAXIMUM PHASE:   4H 9M UT
 PENUMBRAL MAG:    2.366
 UMBRAL MAG:       1.407
 SEMIDURATIONS --
  PNUMBRA:          159M
  UMBRA:            103M
  TOTALITY:        42M
```

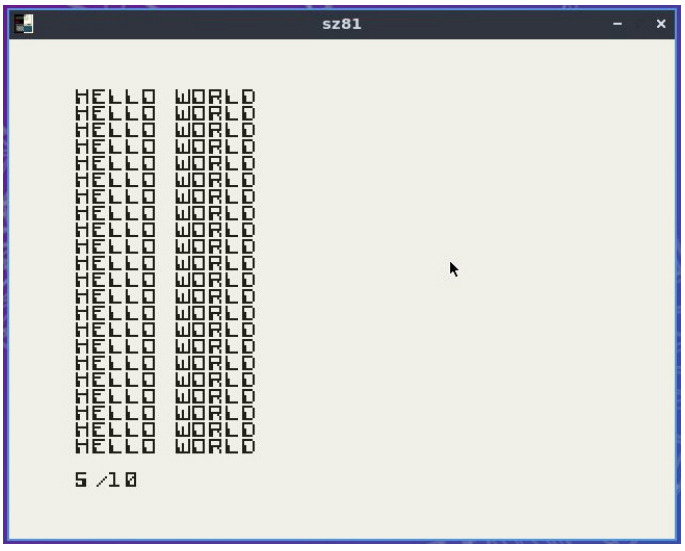
Letting the program run, the next eclipse will be on November 8 at just before 11 am UT.

From the article on the program, the output is "the date and time of maximum eclipse, the magnitude into the penumbra and umbra if that occurs, the semiduration times and length of totality if that occurs. Magnitudes are in lunar diameters into the shadow zones and semiduration times are the times from first contact with the shadow zone to maximum or from max to last contact."

ZX81 Fonts

After tinkering with fonts on the T/S 2068, I was wondering if something could be done on the ZX81 to change its fonts. The ZX81 does not have a way to define a new font, so another method would have to be tried.. Since I am using emulators, the ZX81 ROM is really nothing more than a binary file. If I know where the character definitions are stored in the ROM file, I could use a hex editor to replace these character definitions with ones of my own.

After some digging, I found that the character definitions are stored in locations 1E00 to 1FFF. To keep it simple my first test was to change the 2nd character in the table, CHR$(1) which is one of the graphics characters. To make sure I knew what the characters in ROM looked like (and their values), I created a zxgetfont.bas program that would read the characters from ROM, output the values and show the character. This would allow me to make sure I was editing the right values in the ROM, since I knew what the original values should be.



I quickly created my own UDG character, a bit of a small vertical rectangle. I used "hexedit" in Linux to open the ROM file, go to the right location and change the original values to my new character.

With sz81, you can't change ROMs in the configuration, so I had to save to original ROM file, zx81.rom, to another file name, zx81.rom_orig. I then copies over my new ROM file to zx81.rom. I started sz81 and entered:

PRINT CHR$(1)

and there was my new character. It worked.

The next step was to replace the numbers and letters in the ROM with a new font. Using the fonts that I found for the T/S 2068, I picked the "Digital" font.

The font came with the new characters in decimal, but I needed them in hexadecimal. I changed the name of the font file from "digital.asm" to "digital.csv" and loaded it into a spreadsheet. Since a CSV is a comma separated file, the individual values went into their own cells. Now I used a dec2hex function to convert the values. I know which ones I wanted, so I added a column and marked the rows that were the numbers and the letters.

I copied zx81.rom_orig to zx81.rom_dig for the new font. Using the hex editor, I found the location of zero (0) which is the first of the numbers. I carefully entered in all of the new values for the numbers and then the letters (A-Z). I knew I had it worked out right, as the last character in the ROM table was Z and I was on Z when I got to the last 8 bytes in the table. A quick save of my edits and I was ready to test.

I copied the zx81.rom_dig to zx81.rom, started sz81 and I now have a whole new font.
For each new font you might want, you will need to have a copy of the ZX81 rom modified with that font.

If other emulator read the ZX81 ROM from a file, then this process should work with those emulators.